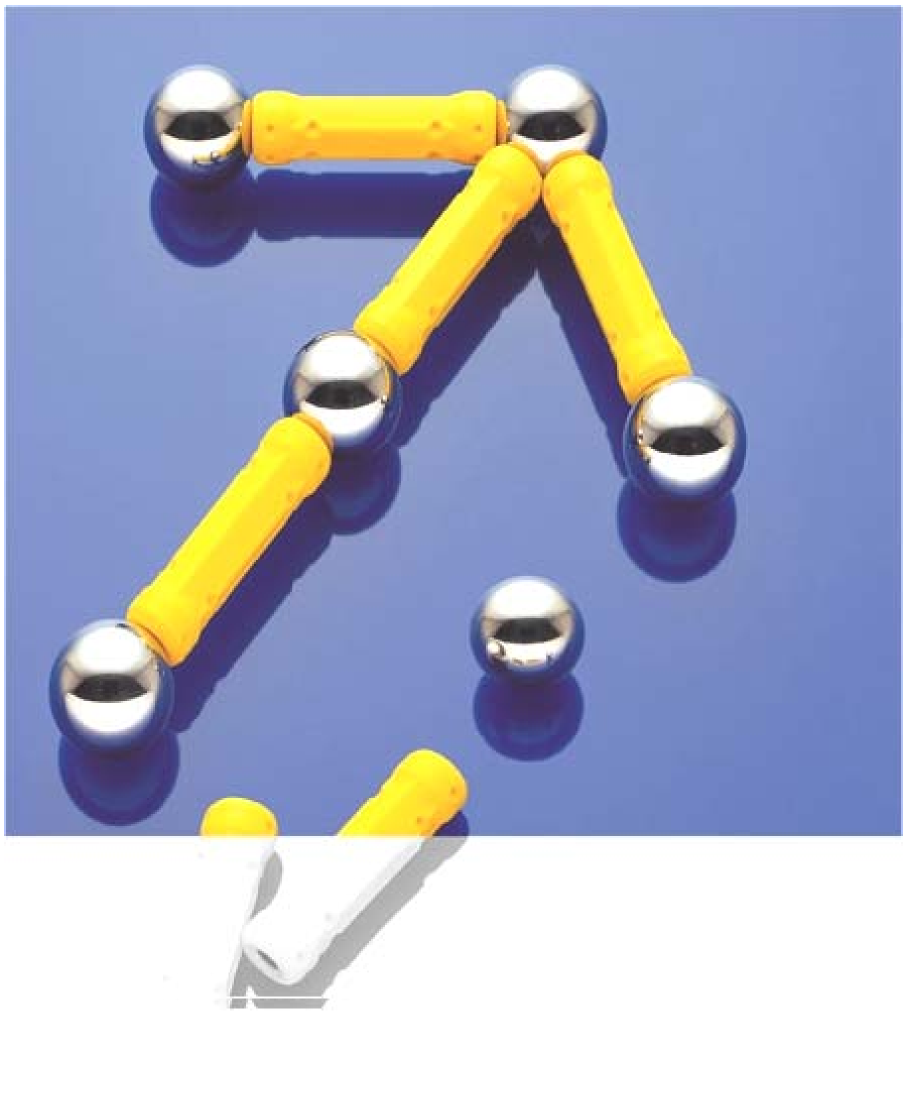
SIEMENS

SOFTNET S7 LINUX/UNIX

Version 4.04



Industrial Solutions and Services
Your Success is our Goal

Contents

1 General Information ... 4

1.1 Structure of the data medium ... 6

1.2 Software packages ... 7

1.3 Documentation ... 8

1.4 Communication protocol ... 9

2 System Requirements ... 10

3 Installation of SOFTNET S7 ... 12

4 Application Start-Up ... 14

4.1 Link instructions ... 15

4.2 Differences Compared with the S7 Programming Interface ... 16

4.3 Additional function “s7_getfds()” ... 18

4.4 Diagnostic data ... 21

4.5 The Transport Name Service TNS ... 25

4.6 TNS Configuration for RFC1006 via TCP/IP ... 27

4.7 Restrictions for Names ... 31

4.8 S7 Connections with the SAPI-S7 Protocol ... 32

4.9 Configuring the VFDs and the Assigned Connection Name ... 34

4.10 Notes on Configuration ... 40

5 TP4 for OSI Communication Protocol ... 41

6 Error Diagnostics ... 42

6.1 Error Diagnostics for Communication ... 43

7 Registration ... 48

General Information 1

Note on installation

The software packages are stored in rpm format on the data carrier and can be installed using the Linux tool rpm or the software manager.

Important rpm commands (Linux)

`rpm –e <package-name>`	Uninstall package
`rpm –ihv <package-name>`	Install package
`rpm –qi <package-name>`	Output package information
`rpm –qa |grep <package-name>`	Check that the package is installed

Important pkg commands (SUN Solaris)

`pkgrm <package-name>`	Uninstall package
`pkgadd -d <package-name>`	Install package
`pkginfo -l <package-name>`	Output package information
`pkginfo |grep <package-name>`	Check that the package is installed

Note on documentation

The manuals are divided into three groups:

- Programming manuals
  The software products provide one or more programming interfaces that are valid for the whole system. For each programming interface, there is a manual describing the function calls with their parameters and dynamic behavior.
- User manuals
  For each software product, there is a user manual describing how to use the product. The user manual contains information about the configuration, start-up, operation, testing tools, and general dependencies of the product with U-NIX/LINUX operating systems.
- Product information
  The product information is this document. It describes the dependencies of the software packages, provides information about the hardware requirements as well as installation instructions.

Note on using the software

The software packages provide you with the programming interfaces in the form of C libraries. These libraries are connected with user programs to make use of the communication services.

The applications communicate with partner applications that run on remote systems and are networked via Ethernet. Programming is independent of the network configuration because the applications can address their partners via freely selectable communication or application relation names.
Before application start-up, the real addresses must be defined for the application relation names. This process, which is often called configuration, and further preconditions for operation are described in the "Application start-up" sections of each user manual.

Note on programming

- The function calls and the dynamic behavior of the programming interfaces are described in detail in the relevant programming manuals
- Deviations from this that are identical for all UNIX/LINUX operating systems are described in the "Creation of applications" sections of each user manual.
- The development tools and therefore also the compile and link options are determined by the particular UNIX/LINUX operating system. The link information and compiler options for Linux are described in this document.

1.1 Structure of the data medium

Note

The CD contains 2 folders:

sw :	This contains the packages to be installed
doc:	This contains the manuals and programming interface descriptions

1.2 Software packages

Overview

Here is a list of the packages provided on the CD:

● **S7**	C programming interface permitting simple and flexible access to SIMATIC S7 system components.
● **PCMX**:	Transport interface to the LINUX driver for communication via the ISO transport protocol. The CMX package also contains the associated configuration software.
● **TP4 (only for ISO8073)**:	Driver for processing the layer 4 transport protocols and the layer 3 network protocol according to the ISO standard.
● **cpitps-redist**	license management.

Note on the TP4 package

TP4 need only be installed for ISO8073 communication. Not for TCP/RFC1006.
For more information see chapter 5.

1.3 Documentation

Product information

Besides the product information this document also contains the installation manual and the user manual.

Manual programming interface

SAPI-S7 programming interface

1.4 Communication protocol

SAPI-S7

Via the SAPI-S7 interface it is possible to use the services of the application layer (layer 7) for communication with the SIMATIC S7. The S7 protocol is used, which is compatible with the ISO transport protocol and TCP/IP (RFC1006).

System Requirements 2

OS requirements:

Linux:

- Kernel version 2.6.x
- Network support (TCP/IP, UNIX-Sockets)
- shared-library support
- shared-memory and semaphore support
- glibc version 2.4 and higher
- POSIX-threads support

SUN Solaris:

Solaris 10

Communication board

A precondition for operation of SOFTBUS is that your system has a correctly installed Ethernet card. You can check that installation is correct by operating the card with TCP/IP. Please consult the Linux documentation to see which Ethernet cards are currently supported by the operating system.

Parallel operation

Communication with the SIMATIC systems is performed using the protocols ISO transport and/or TCP/IP (RFC1006).

Whereas software package CCP-TP4 includes an implementation for communication via ISO transport , the SIMATIC-NET software products use the operating system's implementation for TCP/IP communication.

Parallel operation of TCP/IP and CCP-TP4 is possible via one Ethernet card.

When using the ISO transport protocol, CCP-TP4 supports work with more than one Ethernet connection on each computer system.

Installation of SOFTNET S7 3

Installation of the software is based on the general installation procedures that apply to the target system in question. For the system administrator's tasks each computer contains an administration program with which installation and uninstallation can be performed. Installation can be menu-guided or by command input at the shell level.

Before installation please note the following points:

Integrate the correct driver for your Ethernet board into the LINUX kernel, if you have not already done so, and test the module with TCP/IP.

Step 1: Installation

Please follow this sequence:

1. cpitps-redist
2. Lis-dummy
3. pcmx
4. s7h1

Step 2 Reboot

Boot the system and make sure that the tnsxd process is running. This can be checked with the following command:

ps –ax | *grep tnsxd*

If this is not the case, start the process with the following command:

Linux:

```
/etc/init.d/cmx start
```

SUN Solaris:

```
svcadm enable cmx
```

Step 3 Test

Directory /usr/share/siemens/s7/example contains a simple test program for checking successful installation.

You will find a precise description of handling in the manuals supplied on the CD:

1. SOFTNET S7 Manual
2. SAPI-S7 Programming Interface

Application Start-Up 4

General

SOFTNET-S7/UNIX applications use the functions of the S7 programming interface for communication with remote systems (for example S7 programmable controllers). Communication is based on the following concept: The S7 programming interface uses the term VFD (Virtual Field Device) (see also /1/ “S7 Programming Interface"). An S7 application logs on at one or more VFDs. A list of connections is available for the logon at a VFD. On the S7 programming interface, freely selectable connection names are used. Each individual connection name describes exactly one S7 connection between two end systems. This connection name is used to access the required address information contained in the transport name service (TNS) of CMX. The assignment of the connection names to the actual address parameters of the partner applications is then made while the program is running.

To operate SOFTNET-S7/UNIX applications, two types of configuration file are therefore required:

- Configuration files containing a list of configured VFD names. These configuration files also contain the connection names assigned to a VFD name.
- Configuration files that contain the physical addresses of the S7 connection names: these are in files containing the TNS entries. Using these files, the address information is written to the transport name service and then can be accessed by a SOFTNET-S7/UNIX application while it is running.

4.1 Link instructions

Applications under Linux:

SOFTNET S7/LINUX applications based on the SAPI-S7 interface use the following compiler/link instructions and libraries:

```
LIBS = /usr/lib/libs7h1.so    \
       /usr/lib/libsci_cmx.so \
       /usr/lib/libcmx.so         \
       /usr/lib/libsocket.so  \
       /usr/lib/libkiss.so  \
       /usr/lib/libnsl.so
```

4.2 Differences Compared with the S7 Programming Interface

The S7 programming interface is defined without reference to a particular system and is described in detail in the “S7 Programming Interface" manual. For more detailed information, refer to this manual. SOFTNET-S7/UNIX differs from the description in the “S7 Programming Interface" manual in the points listed below. For details of other supplementary functions or restrictions, please refer to the SOFTNET-S7/UNIX product information for the operating system you are using.

s7_init()

The name of the CP-specific file from which the configuration information is read during the logon, does not end as described in the “S7 Programming Interface" manual with the extension “.LDB". Here, the extension “.dat" is used. Refer also to Section 3.7 of this document “Configuring VFDs and the Connection Names Assigned to Them" in which the configuration of the CP-specific configuration files is described.

The name of the CP-specific configuration file can be assigned freely using environment variables. These environment variables are also described in Section 3.7 of this document “Configuring VFDs and the Connection Names Assigned to Them".

Important

If you use more than one VFD in one or more communications processors at the same time, please read Section TODO of this document completely “Application Start-Up", in particular Section TODO “Notes on Configuration".

Byte Alignment

The SAPIS7 programming interface can only operate when the variables are stored bytealigned in memory. This means that no padding bytes can be inserted, for example between individual components of a structure. This is achieved by appropriate compiler options or by pragmas.

Network Representation of the Variable Values

The SAPIS7 library supplies the read variable values and expects the variable values to be written in network representation (see also the “S7 Programming Interface" manual Section 7.2 “Representation of S7 Variables".
Depending on the host CPU you are using, the variable values must be converted from the host to the network representation before they are sent and from the network to the host representation after they are received.

Block-oriented Services

Functions s7_bsend_req, s7_brcv_init, s7_brcv_stop:
The parameter „r_id“ has a range of values from 1 to FFFF FFFF and not from 0 to FFFF FFFF

Message Services

Function s7_msg_initiate_req:
The function code S7_ALARM_S_ALL_INITIATE was added to the range of possible codes for parameter fct_code.
Function s7_get_msg_abort_req:
The function code S7_ALARM_S_ALL_ABORT was added to the range of possible codes for parameter fct_code.

4.3 Additional function “s7_getfds()”

New S7 function s7_getfds()

The s7_getfds() call returns the file descriptors used by the S7 programming interface. These can be used in the poll() system call. This allows user programs to wait for further external events in addition to communications events. If the poll() system call recognizes events on file descriptors, s7_receive() must then be called. Remember that the s7_receive() call can return S7_NO_MSG.

The file descriptor assigned to an S7 connection is only known after the S7 connection has been established. This means that poll() can only be used to receive events on S7 connections that are already established at the time of the s7_getfds() call (in other words the file descriptor is known).

The file descriptors used by the S7 programming interface must be checked before every poll() since they can change dynamically.

Prototype

```
int s7_getfds (struct pollfd **poll_fdes, int nfdes)
```

Parameter

poll_fdes In **poll_fdes**, the caller transfers a pointer containing a pointer to a structure **poll_fdes**. After successful execution, the caller receives a pointer to an array of structures of the type pollfd. This array contains all the file descriptors currently being used by SOFTNET-S7/UNIX. The array contains an element of the pollfd structure for every file descriptor used by SOFTNET-S7/UNIX. The number of elements of this array is returned in nfdes.
The pollfd structure contains the following elements:

```
int fd;                /* file descriptor */
short events;          /* requested events */
short revents;         /* returned events */
```

For further information, refer to the UNIX Programmer’s Reference Manual description of the UNIX system call poll().

nfdes Pointer to an integer variable.
After successful execution, this integer variable contains the number of elements in the poll_fdes array. The number of elements in the poll_fdes array corresponds to the number of file descriptors currently being used by SOFTNETS7/ UNIX.

Return

0:	Call is ok: the file descriptors have not changed since the last call
1:	Call is ok: the file descriptors have changed since the last call.
-1:	Error occurred, repeat the call, if necessary several times.

If you do not want the user program to wait for additional events, the return values of s7_getfds() can be transferred directly to the poll() system call.

Example of s7_getfds()

```
struct pollfd *fdes;
int nfdes;
int timeout = 5000; /*5 Sekunden */
int ret;
...
if (s7_getfds (&fdes, &nfdes) >= 0)
{
    /* preparation before calling poll */
    ...
    ret = poll(fdes, nfdes, timeout);
    if (ret > 0)
    {
        /* event received */
        /* call s7_receive */
        ...
    }
    else if (ret = 0)
    {
    /* timeout when calling poll */
    ...
    }
    else
    {
        /* poll returned an error */
        ...
    }
}
...
```

For a detailed description of the system call poll(), please refer to the UNIX Programmer´s Reference Manual.

In the example above, additional file descriptions may be contained in the fdes array after the s7_getfds() call and before the poll() call with which, for example, entries made by a user on the user interface can be received.

With s7_getfds() and poll() you can therefore implement a common waiting point within a UNIX process for all events that can be received in the particular UNIX process.

4.4 Diagnostic data

Two new functions were added to the SAPI-S7 to allow access to the system state list SZL. The system state list describes the current state of the automation system.

Function s7_szl_read_req()

With the 's7_szl_read_req()' call, a client application can read SZL data on the server. The variables can only be accessed using their SZL-ID and data set index. The data are transferred to the client in the confirmation from the server and entered in the user buffer by the corresponding processing function ('s7_get_szl_read_cnf()').

Prototype

```
int32 s7_szl_read_req (
                 ord32 cp_descr,
                 ord16 cref,
                 ord16 orderid,
                 struct S7_SZL_READ_DATA *szl_read_data_ptr,
                 ord16 buf_len,
                 void *buffer)
```

Parameter

cp_descr	Handle as return value of the ‘s7_init()’ call.
cref	Reference of the S7 connection which the job will be sent.
orderid	Job identifier to identify the job to be sent and the corresponding confirmation.
szl_read_data_ptr	Pointer to a buffer provided by the user program fort he following structure: `struct S7_SZL_READ_DATA { ord16 szl_id; ord16 index; };` The `szl_id` parameter identifies the system state list. The `index` parameter identifies the data set within the system state list.
buf_len	Length of buffer provided by the user program which is used to store the response data.

buffer	pointer to a buffer provided by the user program which is used to store the response data.

Return values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.
S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Function s7_get_szl_read_cnf()

The 's7_get_read_cnf()' call receives the SZL data that has be read.

With the 's7_receive()' call, the user program receives the 'S7_SZL_READ_CNF' confirmation if the read job could be executed. Following this, the corresponding processing function 's7_get_szl_read_cnf()' must be called for internal processing in the library so that the values can be copied to the user buffer.

Prototype

```
int32 s7_get_szl_read_cnf (
                  ord16 *buf_len_ptr)
```

buf_len_ptr	Pointer to a variable which is provided by the user program. It will be used to return the number of bytes written to the user buffer.

Return values

S7_OK	The function was processed without errors.
S7_ERR_RETRY	This value indicates that an error occurred executing the requested service. This is a temporary problem such as a brief memory shortage. The call can be repeated without modifying the transferred parameters.

S7_ERR	This value also indicates an error in the execution of the requested service. In this case, however, the error does not allow the service to be repeated. Here, steps must be taken to eliminate the error such as assigning new parameters for the call.

Response data

The data written to the user buffer consist of a header structure an d the data read from the system state list. The header has the following structure:

```
struct S7_SZL_READ_DATA_RSP
{
   ord16 szl_id;
   ord16 index;
   ord16 lengthrecord;
   ord16 countrecords;
};
```

`szl_id`	identifies the system state list of the received data.
`Index`	identifies the requestet data set within the system state list.
`lengthrecord`	length of one requestet data set.
`countrecords`	number of all received data sets.

The received data is byte array behind this header. It has to be split according to the parameters `lengthrecord` and `countrecords` in order to get the requested data.

Example

```
int main(int argc, char *argv[])
{
   ord16 cref = 0, orderid = 0;
   int32 cp_descr = 0, ret = 0, rec_ret = 0, readvar = 0;

   struct S7_SZL_READ_DATA_RSP* hdr_data = NULL;
   struct S7_SZL_READ_DATA szl_info;

   ord16 buf_len = 4095;

   rcvbuffer = (ord8*)malloc(4095);
   hdr_data = (struct S7_SZL_READ_DATA_RSP*) rcvbuffer;
...
   establish connection
...
   szl_info.szl_id = 0x19;
   szl_info.index = 0x00;
   /* request SZL data*/
   ret = s7_szl_read_req(cp_descr, cref, orderid,
                         &szl_info, buf_len, rcvbuffer);
   printf( "s7_szl_read_req =0x%x\n", ret);
   do {
      rec_ret = s7_receive(cp_descr, &cref, &orderid);
      switch (rec_ret) {
         case S7_SZL_READ_CNF:
            /*SZL read confirmation*/
            printf("S7_SZL_READ_CNF\n");

            ret = s7_get_szl_read_cnf(&buf_len);

            if (ret == S7_OK) {
               int16 i = 0, j = 0;
               int16 recordLen = ntohs(hdr_data->lengthrecord);
               int16 recordCnt = ntohs(hdr_data->countrecords);

               char* dataPtr = rcvbuffer + sizeof(struct
                                                   S7_SZL_READ_DATA_RSP);
               ord16 dataLen = buf_len - sizeof(struct
S7_SZL_READ_DATA_RSP);

               printf("szl_id       = 0x%04x index        = 0x%04x\n",
                           ntohs(hdr_data->szl_id), ntohs(hdr_data->index));
               printf("lengthrecord = %d countrecords = %d\n",
                ntohs(hdr_data->lengthrecord), ntohs(hdr_data-
>countrecords));
               for (j = 0; j < recordCnt; j++) {
                  for (i = 0; i < recordLen; i++) {
                     printf("%02x", dataPtr[(j * recordCnt) + i]);
                     if ((i+1)%4 == 0) {
                        printf(" ");
                     }
                  }
                  printf("\n");
               }
               printf("\n");
            }
            break;
         default:
            ...
            break;
      }
   } while (rec_ret != S7_SZL_READ_CNF);
...
   end communication
...
   free(rcvbuffer);
   return 0;
}
```

4.5 The Transport Name Service TNS

Transport Name Service (TNS)

Each network and each transport system demands special address information to be able to address a communications partner. With SOFTNET-S7/UNIX, the **Transport Name Service TNS** is used with which the names and addresses of SOFTNET-S7/UNIX applications can be managed.

TNS reads the address information from a directory, the **TS directory** (Transport System Directory). This TS directory contains the address information of every SOFTNET-S7/UNIX application under a symbolic name known as the **global name**. The TS directory must contain information about all SOFTNET-S7/UNIX applications resident on the local system and about the potential communication partners on remote systems.

A SOFTNET-S7/UNIX application operates on the programming interface using symbolic names for the S7 connections. When, for example, the s7_get_cref() function is called, the parameter “conn_name" is used as the input parameter for a symbolic name of an S7 connection (refer to the “S7 programming interface" manual). The **global name** of an entry in the TS directory corresponds to the symbolic name of an S7 connection. The global name of an S7 connection in the TS directory contains both the local address information and the address information for the remote communications partner.

TNS Daemon tnsxd

With all inquiries to the transport name service TNSX, the TNSX daemon **tnsxd** is the central server process. A SOFTNET-S7/UNIX application accesses the TS directory using the TNSX daemon to obtain the address information belonging to a global name.

This procedure is based on the following principle: a SOFTNET-S7/UNIX application sends an inquiry about a global name to the TNSX daemon. The TNSX daemon then accesses the TS directory, reads the required address information and sends this back to the inquiring SOFTNETS7/ UNIX application.

The TNS daemon tnsxd is started when the operating system is started. It can also be started at any time with the following command:

/opt/etc/tnsxd &

TNS Compiler tnsxcom

Entries in the TS directory are created using the TNSX compiler **tnsxcom** with which they can also be read and modified. The TNS entries made in the TS directory are transferred in the form of a file when the TNSX compiler is called. The TNSX compiler expects the TNS entries in this file in a fixed format. Once the format has been checked, the TNSX compiler compiles the entries into the format of the TS directory and writes the entries to the TS directory.

The TNSX compiler **tnsxcom** is called as follows:

```
/opt/bin/tnsxcom [-D] [-u] [filename]
```

-D	Dump mode: tnsxcom outputs all the TNS entries in the TS directory to the “filename" file. This option excludes the “-u" option.
-u	Update mode: tnsxcom reads the TNS entries from the “filename" file, runs a syntax check and writes syntactically correct entries to the TS directory. Any existing entries are updated This option excludes the “-D" option. Calling tnsxcom with the “-u" option is only possible under the root ID.
filename	Name of the file to which the TNS entries will be written (Option “-D") or from which the TNS entries will be read (Option “-u").

TNS Entries

The TNS entries transferred to the TNS compiler tnsxcom in the input file must have the syntax described below for SOFTNET-S7/UNIX.

The protocol architectures (transport systems) supported by SOFTNETS7/UNIX are used depending on the address format in the TNS entries. The following transport systems are supported by SOFTNET-S7/UNIX:

- OSI transport protocol with inactive network layer (null Internet).

- RFC 1006 via TCP/IP

Note

You find a detailed description of a TNS configuration for the OSI transport protocol in the document ***pi_tp4-linux_V60_76.pdf*** .

4.6 TNS Configuration for RFC1006 via TCP/IP

TNS Entry for RFC1006 via TCP/IP

The format described below for a TNS entry selects RFC1006 via TCP/IP as the transport system for a SOFTNET-S7/UNIX connection.

```
<S7 connection name>\
      TA RFC1006 <ip-addr> PORT <remote port> <remote tsel>
      TSEL RFC1006 <local tsel>
      TSEL LANINET <local port>
```

<S7 connection name>

Symbolic S7 connection name. The entry is made in the TS directory with this global name.

Character string with maximum 30 ASCII characters.

The name must be terminated with the “\" character.

There must be an entry in the TS directory for each symbolic S7 connection name used in your SOFTNET-S7/UNIX application.

TA RFC1006

The keyword TA is fixed. The entries following it in this line contain the address information of the remote SOFTNET-S7/UNIX application identified by this symbolic S7 connection name.

The keyword RFC1006 is fixed. It indicates use of RFC1006 via TCP/IP.

<ip-addr>

This is the Internet address via which the remote SOFTNET-S7/UNIX application can be obtained on the network.

The following format is mandatory for <ip-addr>:

<number>.<number>.<number>.<number> where 0 <= number < 256

PORT <remote port>

This is the port number, the remote SOFTNET-S7/UNIX application is using to wait for incoming connection requests. The keyword PORT is fixed followed by the port number.

<remote tsel>

This is the remote transport selector (also known as the “called TSEL"). The remote SOFTNET-S7/UNIX application is addressed on the network using this transport selector.

The following formats are possible for <remote tsel>:

A’<string>’	<string> character string with a maximum of 8 ASCII characters
X’<string>’	<string> character string with a maximum of 8 hexadecimal digits

TSEL RFC1006

The keyword TSEL is fixed. The entries following the keyword in this line contain the local address information of the symbolic S7 connection name.

This keyword is fixed. It indicates use of RFC1006 via TCP/IP.

<local_tsel>

Indicates the local transport selector (local TSEL). The local SOFTNETS7/ UNIX application can be obtained on the network using this transport selector. With active connection establishment to a remote SOFTNETS7/UNIX application, this transport selector is used as the calling TSEL.

The following formats are possible for <local_tsel>:

A’<string>’	<string> character string with a maximum of 8 ASCII characters
X’<string>’	<string> character string with a maximum of 8 hexadecimal digits

TSEL LANINET

The keyword TSEL is fixed. The entries following the keyword in this line contain the local address information of the symbolic S7 connection name.

The keyword LANINET is fixed. It indicates the configuration of a port number.

<local port>

Indicates a local port number, used by the SOFTNET-S7/UNIX application towait for incoming connection requests.

The following formats are possible for <local port>:

A’<string>’	<string> character string with a maximum of 8 ASCII characters, the string has to contain the used port number.

Note

The TCP port 102 is reserved for applications that allow an RFC1006 implementation. This port is used by RFC1006 applications to receive connection establishment requests from other communication partners (which must be possible with the SAPI-S7 protocol).
This port is known as a privileged port on UNIX systems; in other words it must only be used for applications operated under the root ID. This means that SAPI-S7 applications operating with RFC1006 via TCP/IP must be operated under the root ID.

Client and Server Locally with RFC1006 via TCP/IP

Due to the fact that server services are not yet implemented in SOFTNETS7/UNIX, the operation of a client application with a server application purely locally on one computer is not possible.

Example of a RFC1006 TNS entry

```
VERB1\
       TA    RFC1006 149.202.70.16 PORT 102 X'0102'
       TSEL  RFC1006 X'0101'
       TSEL  LANIINET A'2000'

VERB2\
       TA        RFC1006 149.202.70.17 PORT 102 X'0102'
       TSEL      RFC1006 X'0102'
       TSEL      LANIINET A'2001'
```

The line with the reserved word TSEL contains the calling T selector. The line with the reserved word TA contains the called T selector and the called IP addresses.

Please note that if you link to the SIMATIC S7 via the SAPI-S7 interface, the names for the T selectors can only be stated in hexadecimal format with up to 4 characters. Moreover, the T selectors for the SIMATIC S7 have fixed values (see User Manual “SOFTNET-S7/UNIX”).

Caution

The number of possible links for SOFTNET S7 is limited to 64 (ADVANCED-Version) respectively 8 (LEAN-Version). The limitation depends on the existing SIMATIC hardware, i.e. the number of links can be reduced by the hardware.

Deleting Existing TNS Entries

You delete an existing TNS entry as follows:

- Generate an ASCII file with the following entry:

```
TNSname DEL
```

Where TNSname = the TNS entry to be deleted

- Call the TNS compiler:

```
/opt/bin/tnsxcom -u ASCIIfile
```

This call must be made as superuser.
All the TNS entries in the specified file are removed from the TS directory.

- Check the deletion:

```
/opt/bin/tnsxcom -D ASCIIfile
```

This call does not need to be made as superuser.
The entries remaining in the TS directory are written to the specified file.

4.7 Restrictions for Names

Restrictions for Names in TNS Entries

The **logical name of a TNS entry** can be a maximum of 30 characters long. It must start with a letter or the underscore character, the remaining 29 characters can be letters, underscores or numbers.

The **transport selectors** can be either in ASCII format when preceded by an "A" or in hexadecimal format when preceded by an "X".

When using the **ASCII format**, a maximum of 8 characters are possible for the **transport selector**. Permitted characters are letters, numbers or the underscore character.

When using the **hexadecimal format** for the **transport selector**, an even number of a maximum 16 hexadecimal digits can be used. Hexadecimal numbers are made up of the numbers 0 to 9 and the letter a to f or A to F.

Note

Please note that there may be restrictions on the partner systems that go beyond the restrictions explained here.

4.8 S7 Connections with the SAPI-S7 Protocol

Remote TSAP

TSAP of the S7 partner station

The remote TSAP cannot be freely selected but is fixed mainly by the S7 hardware configuration.

It consists of two groups each with two hexadecimal characters.

First group: Device ID, used internally by the SIMATIC S7 PLC.
Possible IDs:

01 = PG, PC (to be used in Softnet)
02 = OS
03 = Other

Second group: Addressing of the SIMATIC CPU, as follows:

(bit 7..5) = rack
(bit 4..0) = slot

To allow the resources of the S7 CPU to be processed (data blocks, inputs/outputs etc.) with the SAPI-S7 protocol, the position of the CPU in the PLC is specified here.

For an S7-400 with only one rack and the CPU in slot 3, this results in a TSAP of **0103**:

The PC communicates directly with the SIMATIC CPU in rack 0, slot 3.

For all SAPI-S7 connections, the same remote TSAP must always be used, the local TSAPs must be different!

The corresponding TNS entries could be as follows:

```
conn_1\
   TA LANSBKA X'<subnet_id>' <Ethernet_addr.> X'0103'
   TSEL LANSBKA A'loc1'

conn_2\
   TA LANSBKA X'<subnet_id>' <Ethernet_addr.> X'0103'
   TSEL LANSBKA A'loc2'
```

Block-oriented Services

In contrast to the variable services, when using the field-oriented services, the connection must be configured at both ends (i.e. also on the S7 PLC).

Note

STEP 7 proposes TSAPs for the connections, and only some of these can be modified by the user. The configuration on the UNIX system must match up with the S7 end; in other words, the local TSAP on the S7 = remote TSAP on UNIX and vice versa.

4.9 Configuring the VFDs and the Assigned Connection Name

General

Using the s7_init() call, a SOFTNET-S7/UNIX application logs on at a VFD of an underlying communication system. As input parameters, the CP name and the VFD name are specified. With this logon, the SOFTNETS7/UNIX application has a list of connections available that can be used for communication with remote systems.

The following sections describe how the VFD names assigned to a communications processor are stored in configuration files. These configuration files also contain the connection names assigned to a VFD name.

Name Conventions

A configuration file <CP_name>.dat must exist for each communications processor used by a SOFTNET-S7/UNIX application. This contains all the VFDs used and the connection names that can be used by a SOFTNETS7/ UNIX application.

The names of the CPs are as follows:

h1_0	CP name for the first board
h1_1	CP name for the second board
... ...	
h1_x	CP name for the nth board, where x=n-1

Within the SAPIS7 library, the TP4 configuration file /opt/lib/tp4/netconf is read to determine the CP names. The CP names are assigned according to the number of communications processors configured for SOFTNETS7/ UNIX. The order in which the communications boards occur in this file determines the order of the CP names.

The name of the configuration file for the individual boards consists of the CP name and the extension "*.dat*", as follows:

<CP_Name>.dat

As a result, the configuration files for the individual CPs have the following names:

h1_0.dat	configuration file for the first board
h1_1.dat	configuration file for the second board
... ...	
h1_x.dat	configuration file for the nth board, where x=n-1

Location of the Configuration Files <CP_name>.dat

Initially, these configuration files must be located in the current directory from which the SOFTNET-S7/UNIX application is started. Using UNIX environment variables, however, each individual configuration file can be moved to any directory. To do this, the following UNIX environment variables are used:

h1_0_s7h1	environment variable for the configuration file for the first board
h1_1_s7h1	environment variable for the configuration file for the second board
... ...	
h1_x_s7h1	environment variable for the configuration file for the nth board, where x=n-1

Example

If the environment variable h1=0_s7h1 is set as shown in the following example, the configuration file h1_0.dat will be expected in the /usr/tmp directory:

```
csh:     setenv h1_0_s7h1 /usr/tmp/h1.cfg
sh:      h1_0_s7h1=/usr/tmp/h1.cfg; export h1_0_s7h1
```

Structure of the Configuration Files <CP_name>.dat

The connection dependent entries in the configuration files for a specific CP are as follows:

```
appl_assoc_name          =        <s7_conn_name>
vfd_name                 =        <s7_vfd_name>
<empty line>
```

The contents must have the following syntax:

appl_assoc_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following S7 connection name.
<s7_conn_name>	Symbolic S7 connection name. Maximum 30 characters. A TNS entry with the same name must also exist for this symbolic S7 connection name (global TNS name = symbolic S7 connection name).
vfd_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following VFD name.
<s7_vfd_name>	Name of a VFD. Maximum 32 characters. The symbolic S7 connection name in the line before is assigned to this VFD.
<empty line>	An empty line used to separate this entry from the next.

The entries above can be repeated as often as required. However there must be at least one entry under appl_assoc_name and vfd_name.

Additionally there can also be configured general parameters. These parameters are used for all connections configured in the file and therefore should be configured on top of the file.

The following parameters can be configured:

max_pdu_size	=	<pdu_size>
amq_calling	=	<amq_calling>
amq_called	=	<amq_called>

The contents must have the following syntax:

max_pdu_size =	Keyword followed by the “=" character that is interpreted as a delimiter from the following maximum PDU size.
<pdu_size>	Maximum PDU size, unsigned integer, max. 8760.
amq_calling =	Keyword followed by the “=" character that is interpreted as a delimiter from the number of acknowledged jobs can be received at the same time on the connection.
<amq_calling>	Number of acknowledged jobs can be received at the same time. Unsigned integer, max. 32.
amq_called =	Keyword followed by the “=" character that is interpreted as a delimiter from the number of acknowledged jobs can be sent at the same time on the connection.
<amq_called>	Number of acknowledged jobs can be sent at the same time. Unsigned integer, max. 32.

All these entries are optional. With them you overwrite the values in the Mini-DB during the s7_init() call. If you don’t configure these values here then the existing values from the Mini-DB are used during the connection establishment. It ist also possible to change these values using the function s7_mini_db_set() during runtime.

Example of the Configuration File "h1_0.dat"

An example of the configuration file “h1_0.dat" is shown below:

```
appl_assoc_name = V1
vfd_name = VFD1

appl_assoc_name = V2
vfd_name = VFD1
```

```
appl_assoc_name = V3
vfd_name = VFD1

appl_assoc_name = V4
vfd_name = VFD2

appl_assoc_name = V5
vfd_name = VFD2
```

In this example, two VFDs are defined for CP “h1_0": VFD1 and VFD2. For VFD1 there are the 3 connections V1, V2 and V3 defined, for VFD2 there are the 2 connections V4 and V5 defined.

Distributing the Address Information on Two TNS Entries

In Sections 3.2 "TNS Configuration for the OSI Transport Protocol" or 3.3 “TNS Configuration for RFC1006 via TCP/IP", the address information for an S7 connection at the local end (see entry TSEL) and for the remote end (see entry TA) is written in a single TNS entry.

The address information for an S7 connection can also be written in two TNS entries. One TNS entry contains the address information for the local end, the other contains information for the remote end.

The TNS entries are then assigned to an S7 connection in the configuration files for the specific CP <CP_name>.dat.

Structure of the Configuration Files <CP_name>.dat

The entries in the configuration files for the specific CPs are structured as follows:

```
appl_assoc_name        =      <s7_conn_name>
vfd_name               =      <s7_vfd_name>
local_name             =      <local_tns_name>
remote_name            =      <remote_tns_name>
<empty line>
```

The parameters must have the following syntax:

appl_assoc_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following S7 connection name.
<s7_conn_name>	Symbolic S7 connection name. Maximum 30 characters. If the parameters “local_name" and “remote_name" below are specified, the S7 connection name entered in “appl_assoc_name" is not used for the TNS access but rather the names specified in “local_name" and “remote_name".

vfd_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following VFD name.
<s7_vfd_name>	Name of a VFD. Maximum 32 characters. The symbolic S7 connection name in the line before is assigned to this VFD.
local_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following TNS name for the local connection endpoint.
<local_tns_name>	Name of the TNS entry for the local connection endpoint. The assigned TNS entry contains the address information for the local end of the S7 connection (parameter TSEL).
remote_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following TNS name for the remote connection endpoint.
<remote_tns_name>	Name of the TNS entry for the remote connection endpoint. The assigned TNS entry contains the address information for the remote end of the S7 connection (parameter TA).
<empty line>	An empty line used to separate this entry from the next.

The entries above can be repeated as often as required. However there must be at least one entry under appl_assoc_name and vfd_name.

Example of the Configuration File "h1_0.dat"

An example of the configuration file “h1_0.dat" is shown below:

```
appl_assoc_name = V1
vfd_name = VFD1
local_name = local_1
remote_name = remote_0

appl_assoc_name = V2
vfd_name = VFD1
local_name = local_2
remote_name = remote_0

appl_assoc_name = V3
vfd_name = VFD1
local_name = local_3
remote_name = remote_0
```

In the example of the configuration file “h1_0.dat" above, three S7 connections “V1", “V2", and “V3" are entered for the VFD “VFD1". The address information for the local end of these connections is in the TNS entries “local_1", “local_2", and “local_3". All three S7 connections use the TNS entry “remote_0" for the remote end.

Corresponding TNS Configuration

These are the TNS entries for the example above:

```
local_1\
   TSEL    RFC1006 X'0001'
   TSEL    LANINET A'2000'

local_2\
   TSEL    RFC1006 X'0002'
   TSEL    LANINET A'2001'
local_3\
   TSEL    RFC1006 X'0003'
   TSEL    LANINET A'2002'
remote_0\
   TA      RFC1006 149.202.70.16 PORT 102 X'0103'
```

The TNS entries “local_1", “local_2" and “local_3" contain address information for the local end (parameter TSEL).

The TNS entry “remote_0“, which is used by all three S7 connections in the example above, contains the address information for the remote end (parameter TA).

The parameters for the entry TSEL for the address information of the local connection endpoint and the entry TA for the address information of the remote connection endpoint are described in Section 3.2 “TNS Configuration for the OSI Transport Protocol" and in Section 3.3 “TNS Configuration for RFC1006 via TCP/IP".

4.10 Notes on Configuration

Mapping the VFD on the CP

The VFDs that are valid on a CP are defined in the configuration files for the specific CP.

The mapping of the VFD names on the CP is unique, a CP can be assigned several VFDs;
however the VFD can only be assigned to one CP.

Mapping the VFD S7 Connection on a VFD

In addition to VFDs, the configuration files for a specific CP also contain the valid S7 connections.

The mapping of the connection name on the VFD name is unique, this means that a VFD name can be assigned one or more connection names;
however a connection name can only be assigned to one VFD name.

Working with More Than One Ethernet Board

The Ethernet board used to establish a connection to the remote SOFTNET-S7 application is selected in the TNS entries using the <subnet_id> parameter. This means that it is not necessarily the CP in whose configuration file the symbolic S7 connection name is entered that is selected for connection establishment. The Ethernet board is selected only using the <subnet_id> parameter in the corresponding TNS entry.

Recommended Configuration

The concept of assigning VFD and S7 connection names in the configuration files for the specific CPs is part of the PCbased version of the SAPIS7 programming interface under Microsoft Windows.

This concept was adopted for SOFTNET-S7/UNIX to ensure compatibility with the PC version.

If there are no compelling reasons to the contrary, such as compatibility with the PC version, a single configuration file “h1_0.dat" should be used with SOFTNET-S7/UNIX applications even if you are working with more than one Ethernet board. To simplify matters, this one configuration file should contain all the VFDs used and the corresponding connection names. This avoids unnecessary confusion during configuration.

TP4 for OSI Communication Protocol 5

Installation

Please follow this sequence:

1. cpitps-redist
2. LiS-dummy
3. silogd
4. tp4
5. pcmx
6. s7h1

Post Installation:

Information about the administration of the TP4 module as well as the connection configurationcan be found in the document ***pi_tp4-linux_V60_76.pdf*** .

Error Diagnostics 6

The following sections describe the tools and logging mechanisms for diagnosing error groups. The errors that can occur when using SOFTNET-S7/UNIX applications can be divided into two groups, as follows:

Communication Errors

The main causes and effects of communication problems are as follows:

Application associations cannot be established. The most common causes are problems with the underlying transport system (for example transport name service not started).

Individual application associations cannot be established. The cause is usually in the configuration or the limit values of the transport system (for example maximum number of connections) have been exceeded.

Application associations break down during operation. These problems can be caused by longer network failures, incorrect response of the partner or by the application not calling *s7_receive()* often enough.

Problems during the data exchange between applications. The cause is usually an incorrect response on the partner or that *s7_receive()* is not called often enough.

Programming Errors

Common programming errors are as follows:

Incorrect parameters are transferred to the S7 functions.

Rare external events (errors, exceptional situations) have not been taken into account.

The *s7_receive()* S7 function is not called often enough.

6.1 Error Diagnostics for Communication

CCP-TP4 Transport System

The STREAM that forms the protocol stack is maintained by the tp4d background process.

Using the UNIX command ps, you can check whether the tp4d process is active.

The tp4stat command is also available with which you can fetch and analyze statistical information from the transport system. Use the following command:

```
/opt/bin/tp4stat
```

Transport Name Service (TNS)

In the TNS, the background process tnsxd manages the address information of the applications involved in communication as TNS entries in the TNS directory.

With the UNIX command ps you can check whether the tnsxd process is active.

The existing TNS entries of the TNS directory can be read into the ASCII file with the name ASCIIfile using the command

```
tnsxcom -D ASCIIfile
```

and checked with the command

```
tnsxcom -s ASCIIfile
```

CMX Library Trace

The trace of the CMX library is controlled by the environment variable CMXTRACE. By supplying the environment variable with a value, the trace is activated and the scope of the information to be collected is specified.

The trace entries of a process are collected as compact binary data in a dynamically created buffer and periodically written to temporary files. These binary files are edited separately with the cmxl tool. The binary files are saved in the directory /usr/tmp. The file names consist of the prefix CMXLa or CMXMa and the process identification number pid.

cmxl reads the entries generated by the trace from the temporary file. The scope of the analysis is decided by the options selected for cmxl.

CMXTRACE: Controlling the Trace

The options specified in CMXTRACE control the trace. The options s, S, and D determine what is logged. The options p, r control the buffering and (wrap) writing of the file:

CMXTRACE = "[-s] [-S] [-D] [-p fac] [-r wrap] [-f directory]"

-s	The CMX calls, their arguments, the options and user data are logged normally.
-S	The calls, their arguments, the contents of any options, the user data in their full length are logged. The options -s and -S exclude each other.
-D	The calls with additional information about system calls are logged in detail. This option is only available in addition to -s or -S.
-p fac	The decimal digit fac determines the buffer factor. The amount of buffering is determined according to fac * BUFSIZ where BUFSIZ is determined by <stdio.h>. If you specify fac = 0, each trace entry is written to the file immediately (with no buffering). fac = 0..8 Default: fac = 1
-r wrap	The decimal number wrap specifies that after wrap * BUFSIZ bytes (BUFSIZ according to <stdio.h>) logging continues in the second temporary file <directory>/CMXMa<pid>. This second file handles the trace in exactly the same way as CMXLa<pid>. After wrap * BUFSIZ bytes, the trace switches between the two files. Following this switch over the content of the file is overwritten. Default: wrap = 128
-f directory	The trace files are written to the specified directory. Default: directory: /usr/tmp or /var/tmp

cmxl: How to read the Trace

cmxl reads the entries generated by the trace from the temporary file, processes the entries according to the selected options and outputs the result to stdout.

The following options specify which trace entries from the file are processed. It is possible to specify more than one of the values described below per cmxl call. Only the options v and x exclude each other. If no options are specified, cdex is used as the default.

```
cmxl [ -c] [-d] [ -e] [ -t] [ -x] [ -D] file
```

-c	The CMX calls for logging on and off the TS application with CMX and for establishing and terminating the connection are processed.
-d	The CMX calls for data exchange and flow control are processed.
-e	The CMX calls for handling events are processed. -t In addition to logging the error messages, the t_error() calls are processed explicitly. Error messages are always logged even if this option is not specified.
-v	The CMX calls, their arguments, the options and the user data are processed in detail. The extent of processing depends on the options specified for CMXTRACE.
-x	The calls and their arguments are processed without options and user data.
-D	This option selects detailed processing with additional information about system calls.
File	Name of one or more files containing trace entries to be processed.

Example of Activating and Evaluating the CMX Library Trace

Example of a configuration for activating the CMX library trace:

```
csh:    setenv CMXTRACE ”-SD -p 0 -r 64 -f .”
sh:     CMXTRACE=”-SD -p 0 -r 64 -f .”; export CMXTRACE
```

Example of a configuration for editing the trace files:

```
cmxl    -Dv CMXLa<pid>   > file_name
```

It is advisable to redirect the data to a file, otherwise they are output to stdout.

Note

CMX error codes and a brief description can be found in the CMX header file /usr/include/cmx.h.

SOFTNET-S7/UNIX Trace

The trace of the S7 library can be controlled by a total of three environment variables, as follows:

S7_TRACE_SELECT

S7_TRACE_DEPTH

S7_TRACE_TARGET

Using these environment variables, you can set the service classes for which the entries will be made in the trace, the trace depth and the trace target (refer to the sapi_s7.h file).

Sample configuration for activating the CMX library trace:

```
sh:       S7_TRACE_TARGET=1; export S7_TRACE_TARGET
          S7_TRACE_SELECT=65535; export S7_TRACE_SELECT
          S7_TRACE_DEPTH=104; export S7_TRACE_DEPTH

csh:      setenv S7_TRACE_TARGET 1
          setenv S7_TRACE_SELECT 65535
          setenv S7_TRACE_DEPTH 104
```

The existence and values of the environment variables are checked when the trace is initialized. Trace settings made previously are then overwritten.

It is advisable to set the trace using the environment variables listed above and not using mini DB calls. This means that the default values can be overwritten for debugging without modifying the application and having to recompile it.

Testing the Transport Connection (tping)

Communication at the transport layer can be checked with the tping program. The program uses the transport name service.

```
tping [-o tnsname1] [-p tnsname2]
```

-o tnsname	global name for the TNS entry for the logon at the local transport system; if the parameter is not specified, the default tping applies.
-p tnsname2	TNS entry with which it is attempted to establish a transport connection; if the parameter is not specified, tnsname1 is used.

The program logs on at the local transport system with the TNS entry tnsname1 and attempts to establish a transport connection with the TNS entry tnsname2. Three reactions to the connection request are possible:

Connection request accepted
Message from tping:

```
„T_CONF received after <time> seconds.
Connection to remote system established!!!“
```

The connection request was confirmed with Connect Confirm by the partner. This proves that the transport system is functioning and that the parameters of the transport layer are correctly set. tping terminates the connection again with DISCONNECT and logs off at the transport system.

Connection request rejected
Message from tping:

```
„T_DISIN received after <time> seconds... returned code:
<error number>: <error description>“
```

The connection request was rejected with Disconnect by the partner. Communication via the transport system is functioning. If a transport connection to this partner is required, parameters must be adapted to the transport layer.

No reaction
Message from tping:

```
„T_DISIN received after <timeout> seconds... returned code:
<error no.>:  Connection can not be set up
              because partner does not respond to CONRQ“
```

Possible causes include:

- There is not partner in the network with this network address or the partner is not operational.

- The partner is configured so that it does not react to incorrect transport layer parameters (for example TSAP).

The functionality of the transport system can be checked using a LAN analyzer. The transport system is functioning when the LAN analyzer records the appropriate connect request PDU.

Note
The CMX error codes (reason) and a short description can be found in the CMX header file /usr/include/cmx.h.

Note
Errors can be investigated in greater detail with the CMX trace tool.

Registration 6

Requesting your registration key (REG_KEY)

To get the registration key you have to fill in the identification code (IDENT_CODE) into the registration form provided with the product.

After the installation of our products you can find out the identification code with the tool "/usr/bin/get_address".

Installing the registration key

When you got your registration key, you have to insert it into the designated file **„/usr/share/siemens/.license.dat“**.

Note

Insert only the registration key into the file. Don’t use any blanks, tabs or comments. If you have more than one registration keys separate them only with a carriage return.

Further information available from

Siemens AG
I IS IN OC

Werner-von-Siemens-Str. 60
91052 Erlangen
Germany

Phone: +49 (9131) 7-46111
E-mail: it4.industry@siemens.com